



# Chapter 2: Java OOP I



XIANG ZHANG  
javacose@qq.com

# Content

2

- OO Concepts
- Class and Objects
  - Package
  - Field
  - Method
  - Main method
  - Object
  - Construct and Initialization
  - Access Control

# Object Oriented

3

- Object
- Class
- Abstraction
  - Design / Implementation
- Inheritance
  - Common / Special
- Polymorphism
  - Method / Behavior

- Example:

每个人都有自己的房子（类），它就像这样（抽象）。我的房子叫Spark House（对象）。



它是一座普通的房子，在设计中还继承了乔治亚风格（乔治亚风格继承了普通房子的特征，同时拥有屋檐装饰）。你会发现无论你把它当做普通的房子还是当做乔治亚房子，虽然外观不同，但是住起来一样的舒适（多态）。

```
House spark_house = new GeorgianHouse();
```

# OO – Another Example

4

```
public class Bird{  
    public void tweet(){System.out.println("JiuJiu~");}  
    public void fly(Place A, Place B){}  
}
```

```
public class Parrot extends Bird{  
    public void tweet(){System.out.println("Hello~");}  
}
```

.....

```
Bird p = new Parrot();  
p.fly(JLH, SPL);  
p.tweet();
```

# Class and Object

5

- Class

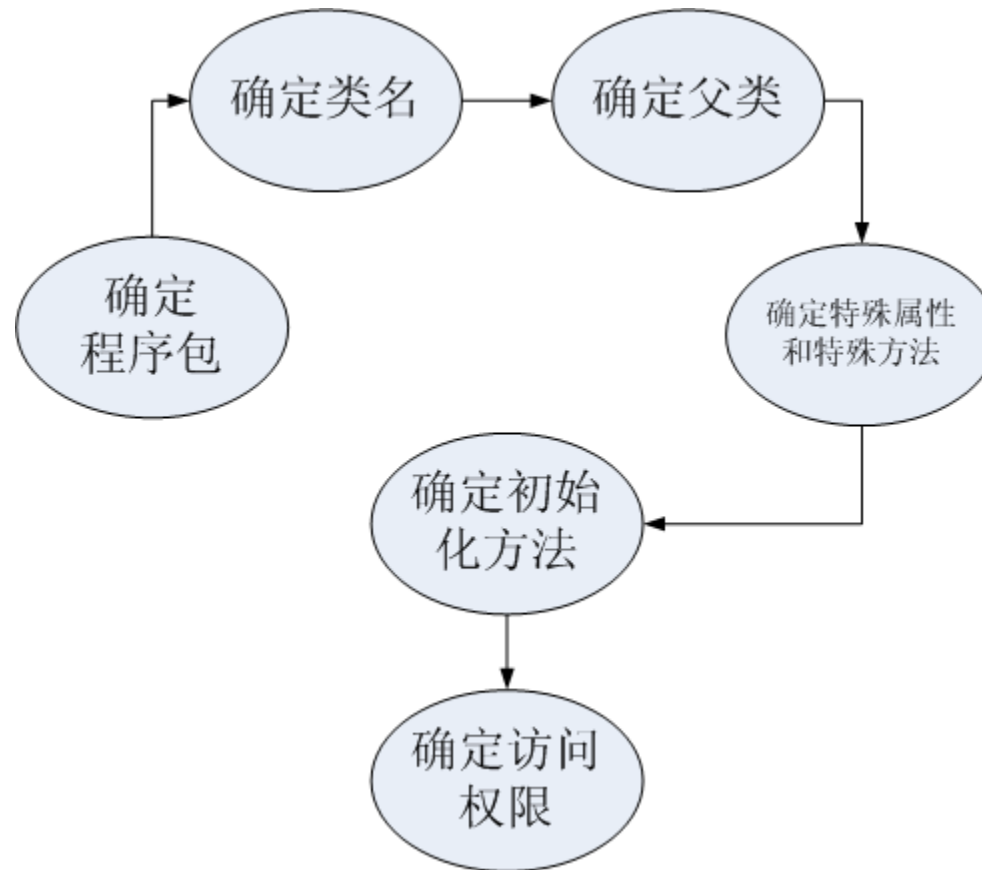
- Often describes a hierarchical concepts, such as: Person、 Bird、 Order
- A class usually has some **attributes** and **behaviors**:
  - ✦ Attributes are called **Fields**, such as the age of a Person
  - ✦ Most attributes character the difference between objects, but some attributes are common, such as each Person has two legs, this kind of attributes is called **Static**
  - ✦ Behaviors are called Methods, and methods can be static too.
- IN Java, the hierarchy of classes is a tree

- Object

- Instances of class. Such as a Person called tom, a octopus called paul...

# Class and Objects – Constructing a Class

6



# A Simple Class

7

```
package cn.edu.seu.cose.javacourse.ch02;
public class Person{
    private String name;
    private int age;
    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }
    public void greet(){
        System.out.println("Hello, I am" + name)
            + " , and I am " + age + " years old.";
    }
    public static void main(String[] args){
        Person tom = new Person("Tom", 18);
        tom.greet();
    }
}
```

```
public class Person {  
  
    private String name;  
    private int age;  
  
    public Person(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
  
    public void greet(){  
        System.out.println("Hello, I am " + name  
            + " , and I am " + age + " years old");  
    }  
  
    public static void main(String[] args){  
        Person tom = new Person("Tom", 0);  
        tom.greet();  
        System.out.println(tom.name);  
    }  
}
```

Is it right?



# Class Components

9

- Package name/ Class name
- import
- Members
  - Field - static / non-static
  - Method - static / non-static
- Access Modifier(Class / Field / Method)
  - public / abstract / final
  - public / protected / private

# Class and Object – Package

10

- Package is a set of Classes
  - To avoid classes with same names
  - To manage classes
- Define a package
  - `package javacourse;`
  - `package cn.edu.seu.cose.javacourse;`
- Import a package or a class
  - `import java.io.*;`
  - `import java.io.File;`

# Class and Object – Field

11

- Define a Field

- Access Modifier
- Static Modifier (Optional )
- Type
- Name

```
public int age;
```

- Non-static

```
public class Person{  
    public int age;  
}  
...  
Person tom = new Person();  
tom.age = 18;  
System.out.println(tom.age);
```

- Static

```
public class Person{  
    public static int counter =0;  
}  
...  
Person tom = new Person();  
Person.counter++;  
System.out.println(Person.counter);
```

*Any bugs?*

```
private String name;
private int age;
public static int counter = 0;

public Person(String name, int age){
    this.name = name;
    this.age = age;
    counter++;
}

public void greet(){
    System.out.println("Hello, I am " + name
        + " , and I am " + age + " years old");
}

public static void main(String[] args){
    for(int i=0; i<10; i++){
        Person tom = new Person("Tom", 0);
        tom.greet();
    }
    System.out.println(Person.counter);
}
```

Any bugs?

# Class and Object – Method

13

- Define a Method

- Access Modifier
- Static Modifier (Optional)
- Return Type
- Name
- Parameter List (Type + Name)
- Method Body

```
public class Person{  
    public int height;  
    // 初始化  
    public boolean isHigh(){  
        if(height>180)  
            return true;  
        else  
            return false;  
    }  
  
    public boolean higherThan(Person someone){  
        if(height>someone.height)  
            return true;  
        else  
            return false;  
    }  
}
```

# Class and Object – Method

14

- Static Method

```
public class Calculator{  
    public static int add(int a, int b){  
        return a+b;  
    }  
}  
...  
System.out.println(Calculator.add(1 + 2));
```

# Class and Object – Overloading

15

- Method Overloading(重载)
  - Method Name
  - Method Signature
    - ✦ Method name
    - ✦ Number of Parameters
    - ✦ Types of Parameters
  - Multiple methods with same name in a class: OK (Overloading)
  - Multiple methods with same signature in a class: **No!**
  - Signature does not include return type, because signature reflects the **specification** of behavior, not the **result** of behavior.

# Class and Object – Overloading

16

- Examples:

```
public String test(String a, int b) {...}  
// a method.
```

```
public void test(String s, int i) {...}  
// Error! Duplicated Methods.
```

```
public String test(int a, String b) {...}  
// Overloading.
```

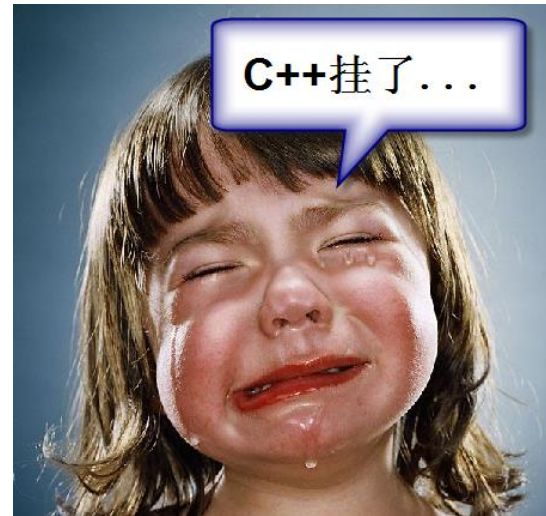
```
public String test(String a, int b, int c) {...}  
// Overloading.
```



# Class and Object – Parameters

17

- Forget them:
  - Formal Parameter?
  - Actual Parameter?
  - Pass by Value?
  - Pass by Reference?
- In Java, the **Copy of Parameter** is passed.
- What is copied?
  - For primary types, their value is copied
  - For objects, the reference is copied. (What is a reference?)



# Class and Object – Parameter

18

- Try:

```
public class ParameterPassing {  
    public static void changeInt(int innerInt){  
        innerInt += 10;  
    }  
    public static void main(String[] args){  
        int i = 5;  
        ParameterPassing.changeInt(i);  
        System.out.println(i);  
    }  
}
```

# Class and Object – Method

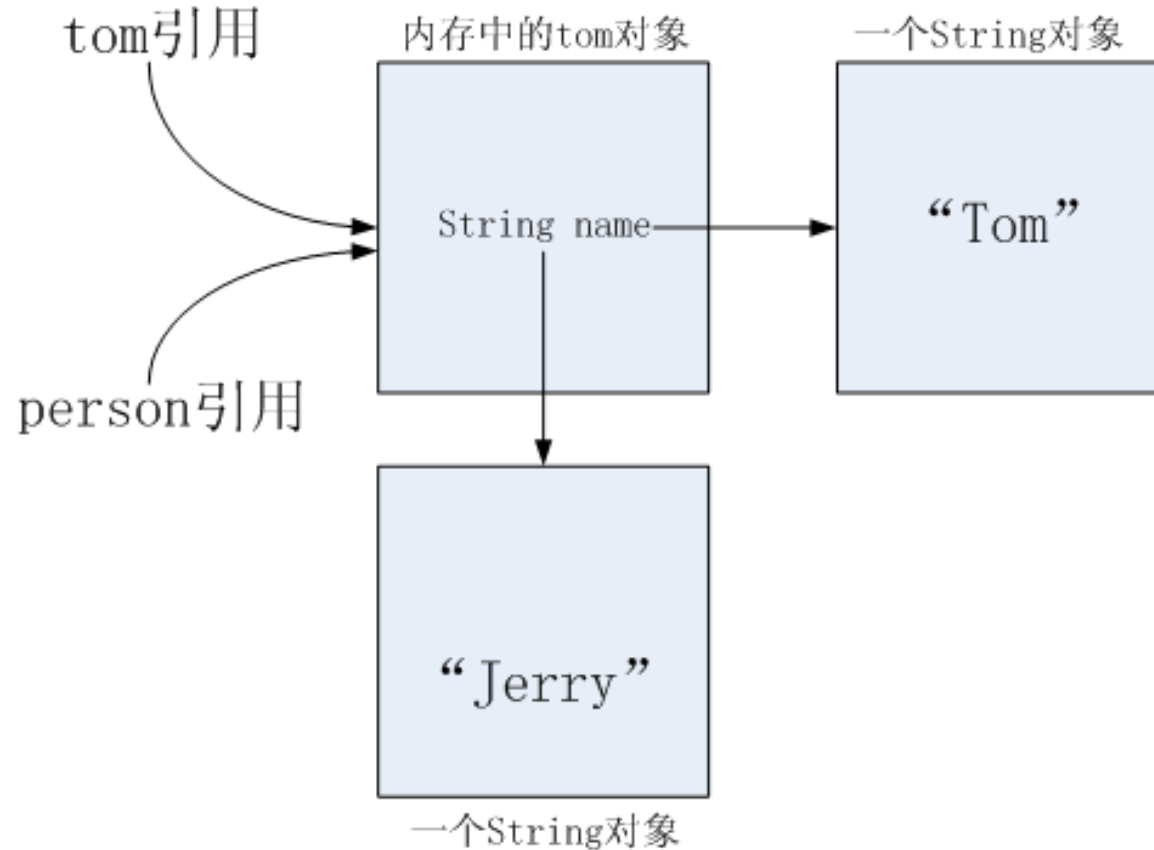
19

- Try again:

```
public class Person{
    public String name;
    public Person(String name){
        this.name = name;
    }
    public static void changeName(Person person){
        person.name = "Jerry";
        person = null;
    }
    public static void main(String[] args){
        Person tom = new Person("Tom");
        Person.changeName(tom);
        System.out.println(tom==null);
        if(tom!=null){
            System.out.println(tom.name);
        }
    }
}
```

# Class and Object – Parameters

20



# Self-teaching

21

- Methods with variable number of parameters

```
public class Calculator{  
    public static int add(int ...numbers){  
        int result = 0;  
        for(int i=0; i<numbers.length; i++){  
            result += numbers[i];  
        }  
        return result;  
    }  
    public static void main(String[] args){  
        System.out.println(Calculator.add(10,11));  
        System.out.println(Calculator.add(10,11,12));  
    }  
}
```

# Class and Object – main method

22

- Each class can have a main method or not
- The main method indicates the entrance of execution
- Each main method looks like this:

```
public static void main(String[] args){  
    ...  
}
```

# Class and Object – Object

23

- Declare a reference of an object, but not create it

```
String s; Person tom;
```

- Declare a reference of an object, and create the object

```
String s = "Hello, World";
```

```
String s = new String("Hello, World");
```

```
Person tom = new Person("Tom", 18);
```

- Null reference: `Person tom = null`

- Security: Reference >> Pointer

# Class and Object – Object

24

- Storage of Objects
  - Registers - inside the processors
  - Stack - object reference, primary types
  - Heap – object themselves
  - Method Area – methods, static data
  - Constant Pool
  - Non-RAM
    - ✦ Streamed Object
    - ✦ Persistent Object



# Class and Object – Object

25

- Destroying Object

- Java GC (Garbage Collection)

- finalize();

- Try:

```
public static void main(String[] args){
    System.gc();
    System.out.println("Memoery: " + Runtime.getRuntime().freeMemory());
    System.out.println("Creating houses...");
    ArrayList<House> area = new ArrayList<House>();
    for(int i=0; i<10; i++){
        area.add(new House());
    }
    System.out.println("Memoery: " + Runtime.getRuntime().freeMemory());
    System.out.println("Colleting garbage...");
    System.gc();
    System.out.println("Memoery: " + Runtime.getRuntime().freeMemory());
}
```

VM arguments:

`-xms100m -Xmx100m`

# Class and Object – Object

27

- All classes in Java inherits java.lang.Object
- All objects in Java have following methods:

```
public boolean equals(Object obj)
```

```
public int hashCode()
```

```
protected Object clone() throws CloneNotSupportedException
```

```
public final Class<?> getClass()
```

```
protected void finalize() throws Throwable
```

```
public String toString()
```

# toString()

28

```
public static void main(String[] args){  
    Person tom = new Person("Tom", 0);  
    System.out.println(tom);  
    tom.greet();  
}
```

- 1) What will happen?
- 2) How to print the name of Tom?

# Reference

29

- Inside The Java Virtual Machine (深入浅出Java虚拟机)

# Construction and Initialization

30

- How to describe the construction of an object in a class?
  - Constructor
    - ✦ Default Constructor
    - ✦ Constructor with parameters
  - Initialization Block

# Construction and Initialization

31

- Constructor

- Default
- With Parameters

```
public class Person(){  
    public String name;  
    public int age;  
    public boolean isEducated;  
    public Person(){  
        this.isEducated = true;  
    }  
    public Person(String name, int age){  
        this();  
        this.name = name; this.age = age;  
    }  
    public Person(String name, int age, boolean isEducated){  
        this(name, age);  
        this.isEducated = isEducated;  
    }  
}
```

# Construction and Initialization

32

- Initialization  
Block

```
public class Person{  
    public int id;  
    public static int counter;  
    {  
        id = counter++;  
    }  
    public static void main(String[] args){  
        Person tom = new Person();  
        Person mike = new Person();  
        System.out.println(tom.id);  
        System.out.println(mike.id);  
    }  
}
```



# Construction and Initialization

33

- Static Initialization Block

```
public class Person{  
    public int id;  
    public static int counter;  
    public static int getBeginID(){  
        ... // 从数据库中获取ID  
    }  
    static{  
        counter = getBeginID();  
    }  
    ...  
}
```

# Construction and Initialization

34

- Think:
  - Why use initialization blocks?
  - What is the difference between initialization blocks and static initialization blocks?

# Class and Object – Access Control

35

- Why Do We Need Access Control?
  - Encapsulation
  - Data Hiding
- Without Access Control:
  - Debugging becomes difficult
  - Data and programs become unsafe

# Class and Object – Access Control

36

- AC Modifier for Classes
  - *default*
  - public
- AC Modifier for Members
  - *default ( package )*
  - public
  - private
  - protected

# Class and Object – Access Control

37

	Same class	Same package	Subclass in different package	Non-Subclass in different package
public	OK	OK	OK	OK
protected	OK	OK	OK	NO
default(package)	OK	OK	NO	NO
private	OK	NO	NO	NO

# Class and Object – Access Control

38

- Getter and Setter Methods

```
...
private String name;
private int age;
public String getName() {return name;}
public void setName(String name) {this.name = name;}
public int getAge() {return age;}
public void setAge(int age) {
    if(age>150 || age<0){
        age = 0;
        System.out.println("Wrong age!");
    }else{
        this.age = age;
    }
}
...
```

# Self-teaching

39

- Data, Information and Knowledge
- Non-structural, semi-structural and structural data
- XML
  - XML and XML Schema
  - XML vs. HTML
  - \* Ant

```
<?xml version="1.0" encoding="utf-8"?>
<Person id="x.zhang">
  <firstname>Xiang</firstname>
  <lastname>Zhang</lastname>
  <organization>
    <College id="COSE">
      College of Software Engineering
    </College>
  </organization>
</Person>
```

# Forecast

40

- Abstraction
  - Abstract Class
  - Interface
- Inheritance
- Polymorphism